Contenido

[----------16/6---------------- 2](#_Toc46913830)

[SUMA 2](#_Toc46913831)

[SIGNO 2](#_Toc46913832)

[SIGNO2 2](#_Toc46913833)

[MAX 3](#_Toc46913834)

[MAX3 3](#_Toc46913835)

[MAX3V2 3](#_Toc46913836)

[BUSCAR 3](#_Toc46913837)

[-----------------23/6---------------- 4](#_Toc46913838)

[MAYOR 4](#_Toc46913839)

[MAXIMO 4](#_Toc46913840)

[POS 5](#_Toc46913841)

[UBICACION 5](#_Toc46913842)

[--------------------29/6------------------ 5](#_Toc46913843)

[INTERCALAR 5](#_Toc46913844)

[------------------30/6-------------------------- 6](#_Toc46913845)

[INTERCALAR ORDENADO 6](#_Toc46913846)

[BUSCAR 6](#_Toc46913847)

[UNIR 6](#_Toc46913848)

[MAX2 7](#_Toc46913849)

[PROUNDIDAD 7](#_Toc46913850)

[INTERCALAR 7](#_Toc46913851)

[ORDENAR 8](#_Toc46913852)

[-------------------3/7------------------ 8](#_Toc46913853)

[UNION2 8](#_Toc46913854)

[CONVERTIR 8](#_Toc46913855)

[---------------------------7/7------------------------ 9](#_Toc46913856)

[INTERSECCION 9](#_Toc46913857)

[POSICIONES PARES 9](#_Toc46913858)

[POS 9](#_Toc46913859)

[USANOD-POS 10](#_Toc46913860)

[SUMOUNO 10](#_Toc46913861)

[VOCALES 10](#_Toc46913862)

[ALUMNOS 11](#_Toc46913863)

[----------------------lunes 13-7-------------------------- 11](#_Toc46913864)

[MASCARA 11](#_Toc46913865)

[;suma recursiva 19](#_Toc46913866)

[--------------------- Ejercicios Javier------------ 21](#_Toc46913867)

## ----------16/6----------------

### SUMA

(defun SUMA (a,b)

(if (or( null a)(null b) nil)

( mapcar ‘+ a b)

)

### SIGNO

(defun sign (x)

(cond

((null x) nil)

((> x 0) 1)

((eq x 0) 0)

((< x 0) -1)

)

)

### SIGNO2

(defun sign2 (x)

(

cond

((eq (sign x) -1)T)

(t nil)

)

)

### MAX

(defun max2 (x y)

(cond ((< x y) y )

(t x)

)

)

### MAX3

(defun max3 (x y z)

(cond ((and(< x y) (< z y)) y)

((and (< x z) (< y z)) z) (t x)

)

)

### MAX3V2

(defun max3v2 (x y z )

(cond ((> x (max2 y z)) x)

(t (max2 y z))

)

)

### BUSCAR

(defun buscar(x l)

(cond

((null l) nil)

((eq x (CAR l)) T)

(t buscarx (cdr l)))

)

)

(setf l '(1 2 3))

(print (max2 2 10))

(print (sign2 -55))

(print (max3 200 10 30))

(print (max3v2 200 10 30))

(print buscar 2 l))

## -----------------23/6----------------

### MAYOR

(defun mayor (x y)

(cond ((> x y) x)

(t y)

)

)

(print (mayor 20 30))

### MAXIMO

(defun maximo (x)

(cond(

( null x) 0

)

(t (mayor( car x) (maximo ( cdr x))))

)

)

(setf x '(20 30 70 10 50 60))

(print (maximo x))

### POS

( defun pos ( a b )

( if ( null a ) nil

( if ( > b 0 ) ( pos ( cdr a) ( - b 1 ) )

( car a)

)

)

)

(setf l '(1 2 3))

(print (pos l 1))

### UBICACION

(defun ubic ( x y )

( if ( or (null x) (null y)) nil

(cons ( pos x (car y) ) ( ubic x (cdr y) ) )

)

)

(print (ubic '(a b c) '(2 0 1)) )

## --------------------29/6------------------

### INTERCALAR

(defun intercalar (x y)

(cond ((and (null x) (null y)) nil)

(t (cons (car x) (intercalar y (cdr x))))))

(print (intercalar '(3 7 8 3) '(2 5 7 9)))

## ------------------30/6--------------------------

### INTERCALAR ORDENADO

(defun intercalar\_ordenado (x y)

(cond

((and (null x) (null y)) nil)

((and (null x) (listp y)) y)

((and (listp x) (null y)) x)

((< (car x) (car y)) (cons (car x) (intercalar\_ordenado y (cdr x))))

((cons (car y) (intercalar\_ordenado x (cdr y))))

)

)

(print (intercalar\_ordenado '(4 7 11 30) '(2 5 8 9)))

### BUSCAR

(defun buscar (x l)

(cond

((null l) nil)

((eq x (CAR l)) x)

(t (buscar x (cdr l)))

)

)

(print (buscar 2 '(1 2 3)))

### UNIR

(defun unir (x y)

(cond ((and (listp x) (listp y)) (append x y))

(t (unir (cdr x) (cdr y)))

)

)

(print (unir '(1 2 3 (2 (6)) 4 5) '(4 5 6)) )

#-----------Faltasacarloepetidos----------------

### MAX2

(defun max2 (x y)

(cond ((< x y) y )

(t x)

)

)

### PROUNDIDAD

(defun prof (x)

(cond

((atom x) 0)

(t (max2( + (prof (car x)) 1) (prof (cdr x))))

)

)

(print (prof '(1 2 ((3)) ) ))

### INTERCALAR

(defun inter ( u w )

( cond

((atom u) w)

((atom w) u)

((< (car u) (car w)) ( cons ( car u) (inter ( cdr u) w )))

(t (cons (car w) (inter (cdr w) u))))

)

(setf l '(3 7 8 3))

(setf h '(2 5 7 9))

(print (inter l h))

### ORDENAR

(defun ordenar (x)

(cond

((null x) nil)

((atom x) (cons x nil))

((inter (ordenar (car x)) (ordenar (cdr x))))

))

(print (ordenar '(1 5 6 10 3)))

## -------------------3/7------------------

### UNION2

(defun union2 (x y)

(cond

((null y) x)

((null x) y)

(t (if (buscar (car y) x)

(union2 x (cdr y))

(cons (car y)(union2 x (cdr y)))

))))

(setf l '(1 2 3 4 5))

(setf h '(9 5 6))

(print (union2 l h) )

### CONVERTIR

(defun convertir (x)

(cond

((eq (car (cdr x)) '++) (list (car x) '= (car x) '+ 1))

((eq (car (cdr x)) '--) (list (car x) '= (car x) '- 1))

))

(setf h '(a ++))

(setf m '(a --))

(print (convertir h) )

(print (convertir m) )

## ---------------------------7/7------------------------

### INTERSECCION

(defun inter (x y)

(cond

((and (null x) (null y)) nil)

((buscar (car y) x) (cons (car y)(inter (cdr y) x)))

((buscar (car x) y) (cons (car x)(inter (cdr x) y)))

)

)

(setf x '(1 2 6 4 5 7))

(setf y '(1 7 5 6))

(print (inter x y))

### POSICIONES PARES

(defun pares (x)

(cond

((null x) nil)

((cons (car x) (pares (cdr(cdr x)))))

)

)

(print (pares x))

### POS

( defun pos ( a b )

( if ( null a ) nil

( if ( > b 0 ) ( pos ( cdr a) ( - b 1 ) )

( car a)

)

))

(setf l '(1 2 3))

(print (pos l 1))

### USANOD-POS

(defun usandopos (x y)

(cond

((null y) nil)

(t(cons (pos x (car y)) (usandopos x (cdr y))))

)

)

(setf x '(1 2 6 4))

(setf y '(1 3 2 1))

(print (usandopos x y))

### SUMOUNO

(defun sumouno (x)

(cond

((null x) nil)

((atom (car x)) (cons (+ (car x) 1) (sumouno (cdr x))))

((listp (car x)) (cons (sumouno(car x))(sumouno(cdr x))))

)

)

(setf x '(1 2 (4 5) 4))

### VOCALES

(defun vocales (x)

(setf voc '(a e i o u))

(cond

( (null x) nil )

( (buscar (car x) voc) (cons (car x) (vocales (cdr x))) )

(T (vocales(cdr x)))

)

)

( print (vocales '(a b c d e)) )

### ALUMNOS

(defun tupla (x m)

(cond

((null x) nil)

((eq (car(cdr(car x))) (car m)) (cons (car x) (tupla (cdr x) m)))

(t (tupla (cdr x) m))

)

)

(print (tupla x m))

## ----------------------lunes 13-7--------------------------

;Una función que aplicada a dos lista de igual longitud, donde la 1era contiene ceros y/o unos y la segunda objetos cualesquiera, devuelva una lista compuesta por aquellos elementos pertenecientes a la 2da lista cuya posición coincide con los valores 1 (uno) de la 1er lista (se utiliza la 1er lista como máscara?¿?¿? )

### MASCARA

(defun binario (bin l)

(cond ((null bin) nil) ((null l) nil)

(t (cond ((eq (car bin) 1) (cons (car l) (binario (cdr bin) (cdr l))))

(t (binario (cdr bin) (cdr l)))

)

)

)

)

(setf bin '(0 1 1 1))

(setf l '(a b c d))

(print (binario bin l))

;9) La función (REVERSE L) devuelve la lista obtenida escribiendo los elementos de L en orden inverso. Por ejemplo, (REVERSE '(A B C)) ---> (C B A).

;Ejercicio: Redefinir la función REVERSE y llamarle N-REVERSE.

(defun init (l)

(cond ((null l) nil)

(t (cond ((> (length l) 1) (cons (car l) (init (cdr l)))))

)

)

)

(setf l '(a b c d))

(print (init l))

(defun n-reverse (l)

(cond ((null l) nil)

(t (append (last l) (n-reverse (init l)))

)

)

)

(setf l '(a b h g c d))

(print (n-reverse l))

;10) Definir la función REV que actúe como REVERSE pero a todos los niveles. Por ejemplo, (REV '(A (B C) (D (E)))) ---> (((E) D) (C B) A)

(defun ultimo (l)

(cond ((null l) nil)

(t (cond ((> (length l) 1) (ultimo (cdr l)))

(t (car l)))

)

)

)

(defun rev2 (l)

(cond ((null l) nil)

(t (cond ( (atom (ultimo l)) (cons (ultimo l) (rev2 (init l))))

(t (cons (rev2 (ultimo l)) (rev2 (init l))))

)

)

)

)

(setf l '(A (B C) (D (E))))

(print (rev2 l))

(setf h '(a b (c d e) j k ))

(print (rev2 h))

;11) La función (REMOVE S L) devuelve la lista obtenida a partir de L borrando S n el primer nivel, Por ejemplo, (REMOVE 'A '(A B (A C) A)) ---> (B (A C)) Ejercicio: Redefinir la función REMOVE y llamarle N-REMOVE

(defun n-remove (s l)

(cond ((null l) nil) ((null s) l)

(t (cond ( (eq (car l) (car s)) (n-remove s (cdr l)))

(t (cons (car l) (n-remove s (cdr l))))

)

)

)

)

(setf h '(A B (A C) A))

(print (n-remove '(a) h))

(defun mismaForma (x y)

(cond

((and (null x) (null y))T)

((and (atom (car x)) (atom(car y))) (mismaForma (cdr x) (cdr y)))

((and (listp (car x)) (listp (car y)) (mismaForma (car x) (car y))) (mismaForma (cdr x) (cdr y)))

(t nil)))

(setf a '(1 ((5)) (3 4)))

(setf b '(1 ((2 6)) (3 4)))

(print (mismaforma a b))

----------------------------------viernes 17-7-------------------------

;13) Definir la función SEPARA que tome una lista (de los niveles que sea) y separe las letras de los números.

;Por ejemplo, (SEPARA '(A (1 2) ((B)) (C (3)))) ---> ((A B C) (1 2 3))

(defun numeros (l)

(cond ((null l)nil)(t (cond ((numberp (car l)) (cons (car l) (numeros (cdr l)))) (t (numeros (cdr l)))))

)

)

(defun simbolos (l)

(cond ((null l)nil)(t (cond ((symbolp (car l)) (cons (car l) (simbolos (cdr l)))) (t (simbolos (cdr l)))))

)

)

(defun nivel (l)

(cond ((null l)nil) ((atom l) l)

(t (cond ((atom (car l)) (append (list (car l)) (nivel (cdr l))))

(t (append (nivel (car l)) (nivel (cdr l)) ))))

)

)

(defun separa (l)

(nivel l)

(cond ((null l)nil)

(t (cons (simbolos (nivel l)) (list(numeros (nivel l))))

)

)

)

(setf h '(A 1 (4 (5) y) A 3))

(print (nivel h))

(print (numeros h))

(print (simbolos h))

(print (separa h))

; 14 ) Definir una función F1 que calcule la media de los cuadrados de las raíces de la ecuación ax2 + bx + c = 0. Por ejemplo, (F1 1 -5 6) ---> 6.5 Nota Existe SQRT

(defun f1 (a b c)

(setq raiz (sqrt (- (\* b b) (\* 4 a c))))

(setq opues (- b))

(setq div (\* 2 a))

(setq r1 (/ (+ opues raiz) div))

(setq r2 (/ (- opues raiz) div))

(/ (+ (\* r1 r1) (\* r2 r2)) 2)

)

(print (F1 1 -5 6))

;15) Escribir una función llamada P4 que calcule el valor del polinomio 3X4 + 7X2 + 4 utilizando la función P2. ( valor del polinomio en orden/grado 2)

(defun p4 (a b c d e x)

(+ (\* a x x x x) (\* b x x x) (\* c x x) (\* d x) e))

(print (p4 3 0 7 0 4 2))

;16) La función (SUBST E1 E2 L) devuelve una lista de L en la que todas las ocurrencias de la expresión E2 se han sustituido por la expresión E1. Ejemplo (SUBST 5 'A '(A B (A C))) ---> (5 B (5 C)) Redefinir la función SUBST y llamarle N-SUBST.

(defun n-subst (e1 e2 l)

(cond ((null l) nil)

(t (cond ((listp (car l)) (n-subst e1 e2 (car l)))

(t (cond ((eq (car l) e2) (cons e1 (n-subst e1 e2 (cdr l))))

(t (cons (car l) (n-subst e1 e2 (cdr l)))

)

)

)

)

)

)

)

(setf e1 '5)

(setf e2 'A)

(setf l '(a b (a c)))

(print (n-subst e1 e2 l))

;17) Escribir la función DOBLA-EL que duplique todos los elementos de la lista que toma por argumento. Por ejemplo, (DOBLA-EL '(A B)) ---> (A A B B) (DOBLA-EL '((A) B)) ---> ((A) (A) B B)

(defun dobla-el (l)

(cond ((null l) nil)

(t (append (list (car l) (car l)) (dobla-el (cdr l)))

)

)

)

(setf l '((a) b c d))

(print (dobla-el l))

;18) Escribir la funci´on DOBLA-ATOMOS que duplique todos los ´atomos de la lista que toma como argumento en el nivel que se encuentre. Por ejemplo, (DOBLA-ATOMOS '((A) B)) ---> ((A A) B B)

(defun dobla-atomos (l)

(cond ((null l) nil)

(t (cond ((atom (car l)) (append (list (car l) (car l)) (dobla-atomos (cdr l))))

(t (cons (dobla-atomos (car l)) (dobla-atomos (cdr l)))

)

)

)

)

)

(setf l '((a) b c d))

(print (dobla-atomos l))

;19 a) Definir los predicados IGUAL-P y DESIGUAL-P

(defun igual-p (n1 n2)

(cond ((eq n1 n2) T)

(t nil)

)

)

(print (igual-p 1 3))

(defun desigual-p (n1 n2)

(cond ((eq n1 n2) nil)

(t T)

)

)

(print (desigual-p 1 3))

;19 b) Definir el predicado MENOR-P tal que

(defun menor-p (n1 n2)

(cond ((< n1 n2) T)

(t nil)

)

)

(print (menor-p 5 3))

(defun suma1 (n1 n2)

(cond ((eq n1 0) n2)

(t ( + (- n1 1) (+ n2 1) ))

)

)

(print (suma1 5 6))

(defun suma2 (n1 n2)

(cond ((eq n1 0) n2)

(t (+ (+( - n1 1) n2) 1) )

)

)

(print (suma2 9 6))

(defun pred (n)

(- n 1)

)

(defun suc (n)

(+ n 1)

)

(print (pred 9))

(print (suc 9))

### ;suma recursiva

(defun suma (n1 n2)

(cond ((eq n1 0) n2)

(t (print (suc ( suma (pred n1) n2 ) )) )

)

)

(print (suma 9 3))

;20 c) Redefinir la función SUMA, SUMA 2, de forma que el número de llamadas a SUMA 2 al calcular (SUMA 2 X Y) sea el mínimo de X e Y. ( usando todo lo definido en 18 y las funciones predecesor y sucesión)

(defun suma2 (n1 n2)

(cond ((= n1 0) n2)

(t (cond ((> n2 n1) (print (suc ( suma2 (pred n1) n2 ) )))

(t (print (suc ( suma2 (pred n2) n1 ) )) )

)

) )

)

(print (suma2 9 3))

;21) Ahora sí redefinir RESTA

(defun resta (n1 n2)

(cond ((< n1 n2) 0)

(t (cond ((= n2 0) n1)

(t ( pred ( resta n1 (pred n2)) )

)

)

)

)

)

(print (resta 9 5))

;22b) Definir la función PRODUCTO realizando otra versión que disminuya el número de llamadas

(defun producto (n1 n2)

(cond ( (eq (or n1 n2) 0) 0 )

(t (suma2 (producto (pred n1) n2) n2) )

)

)

(print (producto 9 5))

;23) Definir los predicados PAR-P e IMPAR-P tales que

(defun par-p (n)

(cond ((eq (mod n 2) 0) T)

(t nil )

)

)

(print (par-p 8))

(defun impar-p (n)

(cond ((eq (mod n 2) 0) nil)

(t T )

)

)

(print (impar-p 9))

;b) realizar lo mismo pero con recursividad CRUZADA

(defun impar2-p (n)

(cond ((par-p n) nil)

(t T )

)

)

(print (impar2-p 8))

(defun par2-p (n)

(cond ((impar-p n) nil)

(t T )

)

)

(print (par2-p 9))

## --------------------- Ejercicios Javier------------

;1. Definir la función suma que devuelva la suma de los elementos de una lista utilizando DO

(defun suma (l)

(DO ((x l (cdr x))

(r 0 (+ r (car x))))

((null x) r)

))

(setf l '(1 2 3 4))

(print (suma l))

;2.Definir la función de factorial utilizando PROG

(defun factorial (n)

(prog ((cont n)

(resultado 1))

etiqueta

(if (= 0 cont) (return result))

(setq result (\* cont result)

cont (- cont 1)

)

(go etiqueta)

)

)

(setf n '4)

(print (factorial n))

;3. Al igual que él punto anterior definir la funcion factorial pero utilizando DOTIMES

(defun factorial(x)

(setf result '1)

(dotimes (cont n result) (setf result (\* result (1+ cont)))

)

)

(setf n '5)

(print (factorial n))

;4) Definir una funcion que eleve al cuadrado los elementos de la lista utilizando la funcion LAMBDA y MAPCAR

(defun cuadrado (l)

(mapcar #'(lambda (n) (\* n n)) l))

(setf h '(1 2 4 5))

(print (cuadrado h))